

APPLICATION  
FOR  
UNITED STATES LETTERS PATENT

10057249-01599  
28563-5425001

TITLE: POLICY IMPLEMENTATION

APPLICANT: SENTHIL PRABAKARAN, DANIEL KIM AND KUL B. SHARMA

CERTIFICATE OF MAILING BY EXPRESS MAIL

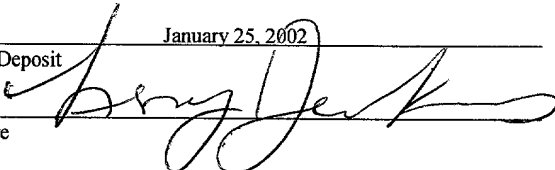
Express Mail Label No. EL624275672US

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

Date of Deposit

January 25, 2002

Signature

  
Leroy Jenkins

Typed or Printed Name of Person Signing Certificate

## POLICY IMPLEMENTATION

### **TECHNICAL FIELD**

This invention relates to policy implementation.

### **BACKGROUND**

5 Policies are a set of enforceable parameters that control the operation and functionality of personal computers and peripheral hardware devices used by the personal computer (e.g., printers). Policies are utilized in both distributed computing environments (e.g., local area networks or wide area networks) and stand-alone personal computers. In a distributed computing environment policies are created and stored in a central computer (e.g.,  
10 a server computer) and downloaded to the individual personal computers linked to the network (e.g., workstation computers) each time a user logs on to the network. In a stand-alone personal computer, policies are created and stored locally on the personal computer.

### **SUMMARY**

15 In an aspect, the invention features a method for providing a network. The network has a first system that generates a request of a policy from the first system to a second system. The second system determines the policy for the first system and provides the policy to the first system.

20 One or more of the following features may also be included. The first system can be a desktop or laptop computer, handheld computer, mobile or desk telephone, personal data assistant, server appliance, numeric or alphanumeric pager, set-top box, air conditioning units, heating units, lights. The second system may be the same as the first or it may be different. The policy managers may be software applications. The data sources may be server-type computers associated with a local-area or wide-area network. The creation and storage of a policy can be facilitated on a separate computer using a plurality of software  
25 applications designed to create policies. All information transfer between the nodes and the policy manager may be done with a markup computer language such as Extensible Markup Language (XML), Directory Services Markup Language (DSML), Simple Object Access Protocol (SOAP), and so forth. The determination of the particular provider needed may be

done using a lookup table based on the policy parameters. The implementation of the policy settings on the particular node requesting said policy may be done in a hierarchical format.

Embodiments of the invention may have one or more of the following advantages.

The technique provides for the management and implementation of computer policies that are applicable to all computers on a heterogeneous network utilizing a plurality of operating systems.

The technique provides a multi-tiered architecture that separates the client from the business logic of policy determination and the specific policy formats and management at the server level.

The technique provides an architecture for implementation of policies on devices that do not have operating systems, i.e., the use of an independent node proxy as part of the multi-tier policy architecture capable of interfacing with non-operating system devices.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

### DESCRIPTION OF DRAWINGS

FIG.1 is an illustration of a three-tier architecture for implementing policies in a network.

FIG. 2 is an illustration of a computer system of a first tier of the three-tier architecture.

FIG. 3 is an illustration of a server system.

FIG. 4 is an illustration of a second server system.

FIG. 5 is an illustration of a first tier of the three-tier architecture.

FIG. 6 is an illustration of a second tier of the three-tier architecture.

FIG. 7 is an illustration of a third tier of the three-tier architecture.

FIG. 8 is an illustration of the steps for implementing policies on a server utilizing the three-tier architecture.

Like reference symbols in the various drawings indicate like elements.

## DETAILED DESCRIPTION

Referring to FIG.1, an exemplary network **10** includes a local area network (LAN) **12** and a local area network (LAN) **14** linked via a bridge **16**. The LAN **12** includes sever systems **18, 20**. The LAN **14** includes computer systems **22, 24** and **26**.

Referring to FIG.2, each computer system, computer systems **22** for example, includes a processor **52** and a memory **54**, memory **54** stores an operating system (o/s) **56** such as Microsoft Windows 2000, UNIX or LINNX, a TCP/IP protocol stack **58**, and machine-executable instructions **60** executed by processor **52** so to perform a client tier policy process **100**, described below.

Referring to FIG. 3, a first selected server system, such as server system **18**, includes a processor **152** and memory **154**. Memory **154** stores an o/s **156**, a TCP/IP protocol stack **158** and machine-executable instructions **160** executed by processor **152** to perform on intermediate tier policy process **200** described below.

Referring to FIG. 4, a second selects server system, such as server system **20**, includes a processor **252** and memory **254**, memory **254** stores an O/S **256**, TCP/IP protocol stack **258** and machine-executable instruction **260** executed by processor **252** to perform a server tier policy process **300** described below.

Referring to FIG. 5, the client tier policy process **100** includes a policy downloading process **102**, a policy parameter formulation process **104**, and application policy handling process **106** and an application event logging process **108**.

The policy downloading process **102** generates a request for download of polices to the server system **16**. Events external to process **100**, such as user logon, computer **50** restart, scheduled download or request for manual refresh of policies triggers the policy downloading process **102**. The policy downloading process **102** interfaces with the policy parameter formulation process **104**.

The policy parameter formulation process **104** calls for each object in the client system **16** that needs to be configured through policies and retrieves state information resident on the server system **16**. In an example, the policy parameter formulator process **104** retrieves state information not specific to a single type of system. Upon retrieving the state information, the policy parameter formulator process **104** packages the state information into a generic markup language format, such as Extensible Markup Language (XML) format,

and sends the packaged information as a request for a policy to a "middle tier system," such as server **116**.

XML is a flexible way to generate common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere. For example, computer makers might agree on a standard or common way to describe the information about a computer product (processor speed, memory size, and so forth) and then describe the product information format with XML. Such a standard way of describing data enables a user to send an intelligent agent (a program) to each computer maker's Web site, gather data, and then make a valid comparison. XML can be used by any individual or group of individuals or companies that want to share information in a consistent way. XML is similar to the language of today's Web pages, the Hypertext Markup Language (HTML). Both XML and HTML contain markup symbols to describe the contents of a page or file. HTML, however, describes the content of a Web page (mainly text and graphic images) only in terms of how it is to be displayed and interacted with. For example, the letter "p" placed within markup tags starts a new paragraph. XML describes the content in terms of what data is being described. For example, the word "phonenum" placed within markup tags could indicate that the data that followed was a phone number. This means that an XML file can be processed purely as data by a program or it can be stored with similar data on another computer or, like an HTML file, that it can be displayed. For example, depending on how the application in the receiving computer wanted to handle the phone number, it could be stored, displayed, or dialed. XML is "extensible" because, unlike HTML, the markup symbols are unlimited and self-defining. XML is actually a simpler and easier-to-use subset of the Standard Generalized Markup Language (SGML), the standard for how to create a document structure.

Referring to FIG. 6, the middle tier policy process **200** includes a policy broker process **202** and a policy provider lookup process **204**. The Policy Broker process **202** is coupled to policy rules **208** resident in memory **154** and the policy provider lookup process **204** is coupled to the policy provider process **206**.

Referring to FIG. 7, the server tier policy process **300** stores policies **310** facilitated by the middle tier policy process **200** from the client tier policy process **100**.

Referring to FIG. 8, the client tier policy process **100** comprises various software components that reside either on a node or node proxy. The Policy Downloader **102** initiates

the download of policies. External events such as user logon, machine restart, scheduled download or request for manual refresh of policies triggers the download process. The Policy Parameter Formulator **104** calls for each object that needs to be configured through policies (node) and retrieves the client state information. In an alternative form, the Policy Parameter Formulator **104** could retrieve information not specific to a single type of node. Upon retrieving the information, the Policy Parameter Formulator **104** packages the information into a generic XML format. The Policy Parameter Formulator **104** sends the packaged information as a request for a policy to the Policy Broker process **202**. The Application Policy Handler **106** reads the final policy contents returned from the Policy Broker process **202** and modifies the configuration of the node. The Application Policy Handler **106** logs all the messages during the process of the policy content to the Application Event Server either directly or through an Application Event Logger **108**.

The Policy Broker process **202** is a middle ware agent that coordinates all communication between the Client and the Data Source and between the different server components. The Policy Broker process **202** gets the request for policies from the Policy Downloader **102** as an XML document of policy parameters. The Policy Broker process **202** then calls the Policy Provider Lookup component **204** and passes the policy parameters. The Policy Provider Lookup component **204** chooses the applicable particular Policy Provider **206** by examining the policy parameters. The Policy Providers **206** are the primary abstraction component to interface with the Directory Service. If there are more than one directory services, each directory service has a corresponding Policy Provider **206**. The Policy Providers **206** each have a unique identification code that is registered with the Policy Provider Lookup Component **204**. The Policy Provider Lookup Component **204** passes the chosen Policy Provider's **206** unique identification code back to the Policy Broker process **202**. The Policy Broker process **202** then invokes a series of Policy Rules **208** that has been registered with it. The Policy Rules Component **208** then modifies the list of policies based on the Policy Parameters or on other custom parameters. The modified list is chained through all the Policy Rules components and returned to the Policy Broker process **202**. After receiving the modified list of policies, the Policy Broker process **202** invokes the Policy Provider **206** and retrieves the content of the individual policies. The Policy Provider **206**

Docket No.: 12849-002001

converts the native policy storage into an XML format. The Policy Broker process **202** returns the content of the policies back to the Policy Downloader **102**.

10057249-01393